High-performance Visualization of UAS Sensor and Image Data with Raster Maps and Topography in 3D

Daniel Stødle, Njål T. Borch, Stian A. Solbø, Rune Storvold

Norut (Northern Research Institute), Tromsø, Norway

Corresponding author email: daniel@norut.no, tel +47 951 64 657.

Mail address: Northern Research Institute c/o Daniel Stødle, Postboks 6434 Forskningsparken, 9294 Tromsø, Norway

High-performance Visualization of UAV Sensor and Image Data with Raster Maps and Topography in 3D

The large amount of data collected during a typical flight with an unmanned aerial vehicle creates challenges for rapid analysis and inspection. A single two-hour flight can collect about 7200 images if the camera captures at 1 Hz, yielding about 20 GB of data. Additional instruments and sensors all add to the size and complexity of the dataset generated by a flight. Efficient visualization of data is one way to deal with the resulting complexity. This paper presents Norut Geo Viz, a system for visualizing data from unmanned aerial vehicles both in realtime and after landing in 3D on a virtual globe. The system has been developed from the ground up to provide control and customizability of the resulting visualization. The architecture, design and implementation of the system is presented, as well as experiences using the system for visualization in the field during flight campaigns conducted by Norut.

Keywords: visualization; 3D; cross-platform; virtual globe; UAV; realtime image mapping

1. Introduction

Unmanned aerial vehicles (UAVs) can collect potentially large amounts of data from onboard sensors and instruments during flight. While airborne, most of the data is typically stored onboard while a small subset of the data is transmitted to the ground control station over available carriers. Existing ground control stations such as Micropilot's Horizon or QGroundControl used by ArduPilot offer limited capability for visualizing flights, typically displaying the UAV's position on a flat map. Further, support for retrieving, verifying and visualizing data from custom onboard instruments is limited or non-existing.

Flight campaigns can be expensive, and it is important for operators to verify that they are acquiring the data they expect. Quickly determining that a given flight will

cover the area of interest and verifying that data is acquired as expected during flight is crucial. For instance, if a given flight is flown at an altitude too high for a laser altimeter to receive reflections from the ground, the entire mission will fail unless the problem is discovered and the flight altitude is adjusted. Similarly, a camera set permanently out of focus or with incorrect exposure time can ruin a mission if not discovered early on. Further, being able to verify that the onboard instruments cover the desired area during flight is essential to discover if in-flight modifications to the patterns being flown need to made.

This paper presents Norut Geo Viz (NGV), a cross-platform system that visualizes sensor and instrument data from UAVs in combination with different raster maps and topographic data in 3D on a virtual globe. The system enables visualization in realtime during flight and allows for rapid data analysis after landing. During flight, NGV fetches a subset of the data collected by the UAV to visualize the UAV's flight path, images captured by onboard cameras, and other instrument data, such as data from spectrometers or laser distance measurements. It also estimates the area covered by the instruments carried by the UAV. The visualization further enables flight planning for autopilots supporting Mavlink (Meier (2009)).

Figure 1 shows an annotated screenshot from the visualization. The UAV is represented by a 3D model of the aircraft. Trailing the aircraft, one or several graphs can be rendered. These graphs represent various sensor or instrument values. Each graph displays the value of an instrument or sensor at the location where the measurement was made, optionally interpolating the values between samples. In the figure, the visualized value is GPS speed. Images captured by the UAV are mapped to the ground based on the camera's intrinsic parameters and estimates the extrinsic

parameters from the UAV's position and attitude at the time of capture. The area covered by the UAV's camera is visualized as the camera view frustum.

The main contribution of this paper is a system architecture and design for flexible, real time and high-performance visualization of flights made by unmanned aircraft. The fully custom design makes it simple to integrate data from existing and new sensors and instruments, enabling operators on the ground to easily visualize that they gather the data they expect during flight.

2. Related Work

There exist a number of systems for 3D visualization of geospatial data. Google Earth (Google, Inc. (2005)) is perhaps the most well-known. Google Earth visualizes the globe using a curated data set consisting of global elevation data, global satellite imagery and several additional layers (roads, weather, buildings, etc.). However, Google Earth is not a tool developed primarily for visualizing realtime data, and as such it is difficult to integrate with the data generated by a typical UAV flight.

NASA WorldWind (NASA (2002)) is an open-source virtual globe visualization API implemented in Java, and designed to be used by developers to create their own virtual globe visualization systems. WorldWind's reliance on Java as its implementation language limits its use to platforms that support Java, which rules out its use on for instance iPhones or iPads without a significant porting effort. gRAID (Jensen (2009)) is a plugin developed for WorldWind. gRAID enables realtime and post-flight overlays of aerial images with imagery rendered by WorldWind, but differs significantly from Norut Geo Viz in that NGV enables visualization of any data collected by the UAV, not just images. Further, gRAID does not make use of the underlying elevation model when projecting images to the ground, whereas NGV utilizes the elevation model when determining image corners in 3D.

WallGlobe (Hagen (2011)) is another system for rendering virtual globes in 3D, developed specifically for high-resolution tiled displays. WallGlobe incorporates some level of realtime data, such as air traffic. However, the system lacks the specialization required to incorporate the varied data sets produced by unmanned aerial flights. Further, the system is closed and not readily available.

3. System Architecture

The CryoWing fleet of UAVs operated by Norut include aircraft based on the Cruiser 1 and Cruiser 2 airframes[1] and the Skywalker X-8 airframe. The CryoWings share a common system architecture, as illustrated in Figure 2.

The autopilot is treated as a black box. Data to and from the autopilot is relayed through the payload control software, which acts as a proxy between the autopilot and the ground station. This decoupling enables the payload to redirect traffic through the radio link that at any given time is the most reliable.

The payload computer is connected to a range of instruments and sensors. Instruments include cameras, infrared cameras, spectrometers and lasers, while sensors include an inertial measurement unit (IMU), a global positioning sensor (GPS), sensors for measuring fuel consumption, power draw and engine revolutions per minute (RPM). During flight, the payload logs data from instruments and sensors, and periodically sends status reports to the ground station. The payload also manages the available communication links. The links used include Iridium satellite modems, 3G/Edge modems, MicroHard radio and WiFi, with typical flights using 1-3 separate links. In addition, the payload provides a web server that can be accessed from the ground either by operators or software. This web server serves logs, status, sensor and instrument data

[1] http://www.etair-uav.com/

upon request. The payload runs Linux on a small form-factor x86 computer in the Cruiser 1 and 2 airframes, and Linux on an Odroid ARM-based computer in the X-8 airframe.

The ground station uses a number of computers to run the necessary ground station software with attached communications hardware. The software consists of four main components: (i) Ground station control; (ii) a database; (iii) a mission data web server and (iv) autopilot software. The ground station control software receives data from the aircraft and logs it to the database. It also responds to dial-ins from the various modems and forwards traffic between the autopilot software and the aircraft. The mission data web server provides access to data stored in the database, and also serves as a proxy to the web server running on the aircraft. This enables actions such as requesting images from the aircraft and caching them locally for visualization or other purposes, without having to utilize bandwidth to fetch the same image from the UAV more than once. It also isolates ground-based clients, such as Norut Geo Viz, from the details of communicating with the aircraft.

The visualization's interface to a given flight is through the mission data web server. The mission data web server serves data from the ground control station during flight, and from data stored in the aircraft's database after landing. The visualization runs on any number of computers or tablets.

4. The Visualization

The visualization is at its core a custom 3D rendering engine tailored for rendering georeferenced data of different kinds: Raster maps and elevation data as well as functionality for incorporating, animating and rendering live or stored flight data, 3D models, images and various custom instrument data.

The visualization relies on several different geographic datasets to effectively visualize a flight. The datasets have either global or regional coverage. The base elevation layer consists of one global and one near-global elevation dataset (U.S. Geological Survey (1996, 2006)), with a global raster layer based on the Blue Marble Next Generation dataset (Stöckl et al. (2005)). Additional global or regional raster and elevation layers are supported. In particular, the visualization currently employs high-resolution elevation and raster data covering Norway and Svalbard from the Norwegian Mapping Authority and the Norwegian Polar Institute. Additional online raster map sources can also easily be incorporated, as well as custom raster layers based on georeferenced satellite or aerial images. A custom data server that handles merging and reprojection of elevation and raster data layers has also been developed as part of the visualization system.

Raster and elevation data is structured as a pyramid with increasingly higher levels of detail (LOD) toward the bottom, as illustrated in Figure 3. The rendering engine represents this structure using a quad-tree scene graph. Each LOD consists of 2^(2n) tiles, where n is the LOD. As the LOD goes up, the resolution of the associated LOD increases. For instance, at LOD zero, the Earth can be represented with a single 256x256 pixel image. At LOD 1, the number of image tiles is 4, for a total resolution of 512x512. An overview of level-of-detail approaches can be found in Luebke et al. (2003).

4.1 Design

Figure 4 shows the visualization's design. The visualization comprises a number of threads with different responsibilities. The main thread is the animation, input and rendering thread. It handles input from the user that moves the visualization's view and otherwise enables users to interact with the visualization's interface, animates objects

and renders the full scene at a target frame rate of 60. When the visualization's view changes, new data becomes visible. To fetch the data, requests are created and passed to the data fetch threads. Once new data is available, the resulting imagery/geometry/flight data is incorporated into the rendering.

The data fetch threads handle requests from other parts of the visualization, typically the main thread. For cacheable data, such as raster map images or elevation data, the request is first checked against a local cache that resides on the file system of the computer or device running the visualization. If the data isn't cached or isn't cacheable, the request is sent to the target server hosting the original data. Data servers include the mission data web server that serves data from a flight, online map servers and the custom elevation and raster data server.

Once a data fetch thread has received data for a given request, the data is cached if possible and passed on to a second set of threads that handle image decompression, geometry construction and data parsing. Raster map tiles and elevation tiles are first decompressed, before they are turned into textures and triangle strip terrain meshes, respectively. The resulting textures and meshes are then uploaded to the GPU. Flight data is parsed and inserted into data structures used by the visualization to keep track of the aircraft and its instruments. By handling these tasks on secondary threads, the visualization system avoids blocking the rendering thread for potentially time consuming tasks and thus improves on the visualization's frame rate.

To reduce memory usage, image and elevation tiles are deallocated from memory when they haven't been used for rendering in the last 0.5-1.5 seconds. The exact threshold varies depending on the target platform. iPads and iPhones have less memory than typical laptops or workstations, and thus require more agressive management of data that isn't currently being used actively by the visualization. The

trade-off is one of network usage and computation versus in-memory caching of already fetched data.

4.2 Implementation

Norut Geo Viz has been developed from scratch in C++ using OpenGL 3.2/OpenGL ES 2 for rendering. At time of writing, NGV runs on Mac OS X 10.8, iOS 6 and Ubuntu Linux 12.04 LTS. Figure 5 shows a photograph of NGV visualizing a flight on an iPad.

4.2.1 Flight data

Flight data both on the aircraft and on the ground station is logged to two separate MySQL databases. Data is served by the mission data web server, which is a small python script that accepts requests over HTTP and performs queries on the local database tables based on the requests it receives. The query results are serialized as JSON before being sent back to the client. The mission data web server also serves images stored in the local file system, potentially requesting the images from the aircraft's web server during live flights. Images can be scaled down before transmission, which is important for reducing bandwidth usage on the link between the aircraft and the ground station during flight.

The visualizer sends requests to the mission data web server to fetch flight data. The system begins by requesting the available timestamp range for a given flight and assigns its current visualized timestamp at the end of the timestamp range. The flight can either be in progress or a stored flight. In either case, the system will poll the mission data web server once per second to determine if the timestamp range has changed. As the visualized timestamp changes, either as a result of user interaction (such as skipping forwards or backwards in the timestamp range) or automatically

(following wall-clock time), the system will request data for a time window of +/-180 seconds around the current visualized timestamp.

The mission data web server responds with data covering the requested timestamp range, which the system then proceeds to parse and insert into a number of interpolators. An interpolator in this context is a map using the timestamp as key and the sensor value as its value, combined with a function for fetching that sensor's value at a given timestamp. Interpolation is used to report a value for a given visualized timestamp. If interpolation is disabled, the closest sensor value for a given timestamp is returned, otherwise linear or cubic interpolation is used.

To map an aerial image to the ground, the visualization gets the aircraft's position, yaw, pitch and roll at the time when the image was taken from the associated interpolators. These values enable the camera's extrinsic parameters to be estimated. Combining the extrinsic parameters with the camera's intrinsic parameters enable construction of a view frustum. Rays from the view frustum's origin[2] towards the view frustum's corners are then intersected with the terrain meshes to determine the latitude, longitude and elevation of each of the image's four corners. These coordinates are used to render the image at the appropriate location on the ground.

A similar approach to that of mapping images to the ground is used to check laser distance measurements. A ray is cast from the aircraft's position and rotated according to the aircraft's yaw, pitch and roll toward the ground. This produces an intersection with one of the terrain meshes and enables calculation of the aircraft's distance from the ground along the same vector that the laser distance measurement

[2] The aircraft's position.

instrument is pointing. Depending on the quality of the underlying elevation data, this enables quick verification of laser distance measurement results.

A rendering of spectrometer data is shown in Figure 6. Spectrometer data is rendered as a set of points. Measurements are rendered at the location where it was made, tilted and oriented according to the aircraft's yaw, pitch and roll.

4.2.2 Image mosaics

The visualization system can output georeferenced image mosaics at arbitrary resolution for any given flight. To build image mosaics quickly and without requiring vast computational resources, the system georeferences flight images using the same approach as used for mapping images to the ground during flight/after landing. The system then uses each image's latitude and longitude corner points to create a 2D bounding box encompassing all images taken during a flight. The user then decides on an output image resolution (in pixels). To create the output image, the system divides the image into 1024x1024 pixel tiles. Each tile is processed in sequence, until the entire image has been constructed. Every image that overlaps a given tile is fetched from the mission data web server, and rendered into an OpenGL framebuffer object (FBO). When a tile is fully rendered, pixels from the tile are read back from the FBO and written to the appropriate part of a tiled TIFF image.

During rendering, images can be filtered based on different criteria. For instance, images with little color variance can be deemed as being 'taken in the clouds' and ignored, or images taken while the aircraft is turning can be ignored based on pitch and roll thresholds. Figure 7 shows a crop from two image mosaics generated during an oil spill exercise conducted near Berlevåg, Norway. The left mosaic has not been filtered for cloud-images, while in the mosaic on the right most cloud-images have been automatically removed.

4.2.3 Input and camera control

The visualization can be controlled either using a mouse and keyboard with 'standard' first-person-gaming controls or using multi-touch gestures on either a trackpad, tablet display or table surface. Figure 8 shows the visualization running on a multi-touch table. The table's operating system is Linux, and uses TUIO (Kaltenbrunner et al. (2005)) to route multi-touch input to the visualization.

A virtual camera abstraction is used to control what the visualization renders and how the rendering is projected. The result is a clipped-pyramid view frustum, with six sides: A near and far plane, as well as planes for the left, right, top and bottom parts of the display. The camera has a position and perpendicular vectors pointing up, left and along the view direction of the camera.

4.2.4 Culling and selection of LOD

The visualization uses view frustum culling and view-dependent level of detail to reduce the amount of geometry and the number of textures that need to be rendered for a given frame. For each frame, starting at the root of the quad-tree scene graph, the system **traverses** the graph recursively, checking the bounding spheres/boxes of each scene node against the current view frustum. Any node whose bounding sphere/box falls outside the view frustum is culled. All descendants of such tiles are also culled, since any child of a scene graph node must be contained in the parent's bounding sphere/box. Any nodes that intersect or are fully contained by the view frustum are added to a 'render set' if they meet the level of detail criteria; otherwise, culling continues another level down in the quad-tree. At the end of this process, a set of tiles to render is produced at their appropriate levels of detail.

The criteria used to select which level of detail to render a particular part of the scene graph at are as follows. First, a minimum level of detail is enforced, ensuring that

the Earth appears as a sphere even when viewed from far away. The second criterion tests whether the current camera position is within an enlarged bounding sphere of the tile being evaluated, and if so increases the level of detail. Finally, the distance from the camera to the tile's center is computed and checked against a look-up table of log-based thresholds indexed by the current node's level of detail. If the distance is above the threshold, the level of detail for that part of the scene graph is increased. In contrast to for instance Google Earth (Google, Inc. (2005)), the level of detail is not selected based on the number of pixels a given tile would occupy on screen if rendered. The reason for this is that the pixel-based approach fails when the camera's view direction is perpendicular to the ground and the camera's position is close to the ground.

4.2.5 Terrain meshes

Figure 9 illustrates a common issue when rendering neighbouring terrain meshes at different levels of detail. Due to the lower LOD used to render the upper mesh, gaps appear between the high-resolution mesh and the low-resolution mesh. A similar issue occurs with meshes at identical LOD, due to precision errors when computing the final position of a given vertex on the GPU.

The visualization handles gaps and cracks in neighbouring meshes after culling, by shifting the positions of edge vertices in the high-resolution mesh to match those of the lower-resolution mesh using linear interpolation. For meshes at identical resolution, edge vertices are copied directly from one of the meshes to the other. The results are stored in video memory until a tile is deallocated. The runtime cost of using the technique is sufficiently low to enable its use even on portable devices like iPads or iPhones without adversely affecting performance.

4.2.6 Stereo rendering and shadows

The visualization supports split-screen stereo rendering, enabling compatible consumer TVs and projectors to display output from the visualization in 'true' 3D and enabling a better sense of depth and distance for users of the system. Stereo rendering is implemented by rendering the scene twice, once for the left eye and once for the right eye (Bourke (1999)). The scene is culled and the level of detail for each tile is selected using the main virtual camera. Two additional virtual cameras are created, offset slightly left and right of the main camera. This is done to avoid selecting different levels of detail for textures and terrain meshes for the two viewpoints. The two renderings are done at half the horizontal resolution of the regular viewport, using the virtual cameras for the left and right eye.

Given a date and time of day, the visualization can simulate the sun's position to cast shadows from terrain and objects onto the ground. Objects on the ground can come from custom 3D models, derived from raster map layers or be imported from Collada or KMZ-models[3], such as those contributed by users to Trimble's 3D warehouse and used by Google Earth. Shadow casting happens in realtime using a technique known as Variance Shadow Mapping (Donnelly & Lauritzen (2006)). The scene is rendered twice for each frame, once from the point-of-view of the sun, and the second time from the virtual camera's viewpoint. A GPU shader uses data from the first rendering pass to do probabilistic tests for whether a given fragment is obscured by a shadow caster. An example of this is shown in Figure 10, where a model of the Tromsø bridge and box building models cast shadows onto the terrain.

---

[3] The Collada-parser was developed from scratch, and does not support the full feature set of the Collada standard. KMZ-files are used to embed Collada 3D models and georeference them.

4.2.7 Spatial jitter

Most current GPUs perform best when doing calculations using 32-bit single-precision floating point values for vertex data. Double-precision vertex data is usually not supported on typical consumer GPUs. When support is available, any benefits gained from using double-precision data are quickly offset by the added cost of transferring and storing twice the amount of data on the GPU. Double-precision data also comes with a computational cost on the GPU-side, resulting in reduced rendering throughput. For GPUs where double-precision floats are not available, such support can be emulated at an additional cost of doing manual conversions in the vertex shader (Göddeke et al. (2005)), however on GPUs employed by smartphones and tablets, memory pressure and bandwidth considerations further argue for keeping data on the GPU in single-precision (or less) format.

The visualization uses double-precision data to store and work with its terrain data, but transforms the data to single-precision floats when transferred to the GPU. During rendering, as the camera's position moves towards a mesh, a phenomenon known as 'spatial jitter' can occur (Thorne (2005)), especially for large scenes. During rendering, a vertex will typically be transformed from object space to world space, before going through the perspective divide to end up in clip space[4]. From clip space, the geometry the vertex contributes to is rasterized to the display. When the camera is close to a mesh, this results in several transformations that gradually exhaust the precision of floating point values resulting in the vertex's final location in clip space.

---

[4] Object space is a coordinate system specific to a given object in a rendering engine. The modelview matrix is used to transform an object space coordinate to its world space equivalent. Clip space is a coordinate system ranging from -1 to 1 in all axes, which the GPU uses to generate fragments and do depth testing. The projection matrix is used to transform the world space coordinate to clip space.

When the resolution of the single-precision floating point values is exhausted, geometry on screen jumps around rather than staying fixed at their expected pixel positions. To overcome this problem, the visualization uses a technique known as ‘relative to center’ rendering (Thorne (2005), Ohlarik (2008)). For each terrain tile to render, the mean 3D position of all vertices constituting the mesh is computed. This value is then subtracted from each vertex, thereby bringing the actual coordinates for each vertex closer to zero (where floating-point precision is at its best). When rendering, the modelview-matrix is configured as normal, except that instead of translating it by the eye position, it is translated by the difference between the eye position and the mesh’s center. This way, as the camera gets closer to a given mesh, that mesh will undergo smaller and smaller translations away from 0 and thus gain additional precision. The same approach is used to render objects, such as the UAV or 3D models of buildings.

4.2.8 Flight planning

Figure 11 shows the flight planning interface in Norut Geo Viz. Waypoints are either read from a file on disk or from a Mavlink-capable autopilot. The visualization enables adjustments to the waypoints based on their elevation above the terrain model, enabling operators to ensure that flights are planned in a way where the UAV will not fly outside the altitude envelope specified by a flight permit. To accomplish this, the visualization analyzes the flight path between waypoints and compares the elevation of the current position along the path with the terrain model. If the elevation above ground is outside the permitted altitude envelope, the waypoints contributing to the current path segment are adjusted iteratively until the entire path fits within the specified altitude envelope. The system uses 2.5-meter increments along the flight path to conduct the analysis. At present no attempt is made to simulate flight characteristics or handle special autopilot

commands.

The shadow casting implementation described earlier is reused for enabling simplified analysis of radio coverage and to display areas that could potentially be in radio shadow. The user positions the antenna somewhere on the terrain model, and targets the antenna by placing a second point somewhere on, above or below the terrain in the direction a given flight is planned. Terrain features between the antenna's position and its target then cast shadows on vertical 'skirts' projected down from the path the UAV is expected to take between waypoints. Using this interface, the user can visually determine where the antenna should be placed to provide radio coverage for a given flight, or alternatively the approximate altitude at which the UAV needs to fly to stay within coverage. While the system doesn't take reflections or other radio wave phenomena into account, it does portray the effects of any obstacles included in the terrain model with reasonable accuracy.

4.2.9 Elevation and Raster Data server

The Elevation and Raster Data (ERD) server provides the visualization with elevation data for rendering the terrain model and raster data stitched together from different sources such as satellite or aerial imagery. The server is a custom, multi-threaded web server handling requests from the visualization, performing computations in response to requests and delivering results either as TIFF files/raw data for elevation requests or as JPEG/PNG for raster requests.

All datasets operated on by the ERD server are described by JSON-encoded property-files. The properties used include a dataset name, the spatial extents of the dataset and the geographical projection the dataset is in, as well as the dataset's maximum level of detail. Datasets may be stored either as flat files in a directory

structure on disk, or in an SQLite database following the MBTiles specification[5]. All elevation tiles are 64x64 pixels in size, and cover the equivalent area of one 256x256 pixel raster tile at the same level detail.

When an elevation tile is requested, the server constructs its response as follows. Data from the global elevation layer is fetched from disk. If additional elevation layers overlap the area requested by the visualization, data from these layers are fetched as well. For instance, a request covering a part of Norway would first use data from the global elevation layer to initialize the response, before data from the higher-detail Norway elevation layer would be used to increase the spatial resolution of the response. This approach ensures that areas where the Norway elevation layer doesn't have valid data, will still be served with valid elevation data from the global elevation layer. If only a single elevation dataset contributed to the response, the source 16-bit grayscale TIFF file is sent unmodified. Otherwise, the server responds with raw elevation data resulting from the merge of the source elevation datasets.

The ERD server also provides merging of raster tiles and reprojection of source data from UTM to the Web Mercator projection. This process is accelerated using OpenGL. For the general case, where reprojection is necessary before a request can be served, the server starts by fetching all source tiles necessary for a given response tile. Data may be fetched from disk (for local datasets) or from other online map servers. Reprojection parameters are obtained using the OGR library[6], and used to obtain the reprojected corner coordinates for a given tile when reprojected from its source projection to the Web Mercator projection. Each required source tile is then rendered into a 256x256 pixel FBO, before the end result is read back from the graphics card. If

---

[5] https://github.com/mapbox/mbtiles-spec (last visited October 15. 2013).

[6] http://www.gdal.org/ogr/ (last visited October 15. 2013)

the result contains translucency, the response is encoded as a PNG; otherwise the result is encoded as JPEG.

5. Evaluation

NGV maps images to ground using a simple first-order approximation of each image's location based on the onboard camera's position and orientation, as determined by the GPS and IMU. To evaluate the accuracy of these mappings, 496 images were post-processed into an image mosaic using AgiSoft's PhotoScan software. Of these, PhotoScan used 283 of the supplied images to generate the image mosaic. For purposes of this evaluation, we assume that the results of creating an image mosaic with PhotoScan provides a "ground truth" to which NGV's accuracy can be compared. This is a simplification, as the camera positions exported by PhotoScan only yield an approximate estimate of each image's final position in an image mosaic, does not account for the elevation model built as part of the image bundling performed by PhotoScan and gives no details as to the weight each image is given in the final output. No ground control points were established for the flight in question.

After processing, camera positions for each image were exported and compared with NGV's mapping of the same 283 images. Figure 12 shows a histogram of distance in meters between camera positions computed by NGV and camera positions computed by PhotoScan. The mean camera position distance is 18.9 meters and the median is 17.5 meters. For both NGV’s camera positions and PhotoScan’s camera positions, five rays were projected to the ground, yielding the corner and center positions of each image. Figure 13 shows the resulting histogram of the distance between corresponding corner and center positions. The mean difference is 88.6 meters, and the median is 55.5 meters.

6. Discussion

Norut Geo Viz is a fully custom visualization system implemented from scratch. The motivation for developing the system from scratch rather than relying on already existing visualization systems is control. Developing from scratch gives control over all implementation choices that in turn affect all aspects of the visualization. It makes it possible to understand the resulting system's performance, and make the appropriate trade-offs for the 'special case' of visualizing UAV flights. Knowing the limits and behaviours of the system is essential when flying missions. This control also translates into simpler ways of tailoring the rendering engine to display different kinds of instrument data. Control of the code and its development also simplified the process of porting the system to run on iPads and iOS, a process that took about a week primarily because the main parts of the system were already developed in platform-independent ways using portable APIs and libraries wherever possible. Making changes in the field to support new features or adjust existing features is also made possible by having control and a deep understanding of the code base.

Norut Geo Viz visualizes flights in near realtime. Due to latency introduced by both the communication link to the aircraft and the 'once-per-second' polling strategy employed to update timestamp ranges, the latency between the aircraft actually being at a specific geographical location and the visualization reflecting the aircraft at that location can be up to approximately 5-6 seconds. The majority of this latency stems from the worst case latency of the communication link (assuming no link drops). We have observed latencies on Iridium links of up to about five seconds. On alternative links the communication latency is significantly lower (on the order of 30-500 ms). An alternative to the polling strategy currently used is to make blocking requests to the

mission data web server, that return once new data is available. This is currently future work.

Figures 14 and 15 show two different images and their footprint on the ground as computed by both PhotoScan and NGV. Figure 14 shows an image mapped to the ground where the discrepancy between PhotoScan and NGV is quite large, with a maximum distance between corresponding corner points of 622.9 meters. The primary reason for this is NGV's inaccurate estimate of the aircraft's attitude, as the corresponding camera position is quite close at 24.3 meters. Figure 15 shows an example of a better match, where the estimate made by NGV is closer to the estimate made by PhotoScan. The maximum distance between corresponding corners for this image is 14.0 meters, with a corresponding camera position distance of 10.2 meters.

The results of mapping images to the ground can be inaccurate, for two primary reasons. First, the timestamps assigned to images taken during flight are inaccurate. The cameras currently used are commodity digital single-lens reflex cameras (DSLRs), with clocks that aren't synchronized to the payload computer's clock. This makes selecting which sample(s) to choose from the IMU[7] challenging. Currently, the images are timestamped with the local time on the payload computer at the moment the image begins transferring from the camera, which doesn't account for time spent capturing and compressing the image on the camera. Second, the result is highly dependent on the aircraft's position and attitude being accurate. Regardless of timestamp accuracy, drift in the IMU or a badly calibrated IMU can result in less than perfect image-to-ground mappings. However, as an approximation the approach used by the visualization is good enough and can produce surprisingly good mappings. In practice the results are best for images taken during extended periods of level flight in one direction.

---

[7] The IMU is an Xsens MTi-G that provides data at 100 Hz.

The same issues also affect the final quality of the image mosaics the visualizer can produce. However, for building a high-resolution image mosaic, the speed at which a mosaic can be made offers distinct benefits over more accurate solutions such as Agisoft's PhotoScan in situations where a rapid result is required -- for instance when mapping oil spills and clean-up crews need information on 'where to go.'

A further weakness of the current image-to-ground mapping is that the resulting image is mapped onto a single quad. Even though each corner of the quad is mapped accurately to the ground's elevation model, there may be peaks, valleys or other features in the elevation data set that are not reflected in the resulting mapping. This could be solved at some computational cost by adding additional vertices to the image mapping based on ray-intersections with the terrain meshes, or by draping the image directly over the elevation mesh. These approaches will be explored in the future.

6.1 Operational experiences

Norut has deployed the visualization in the field during several flight campaigns. In July-August 2012, the system was used for a month-long campaign conducted from Ny-Ålesund on Svalbard. On Svalbard, the aircraft mostly used its Iridium satellite link to communicate with the ground station, resulting in very limited bandwidth (200-500 bytes/second) and very high latencies. However, the visualization still proved useful during flight to observe the aircraft and compare its flight trajectory with the underlying elevation model. After landing the system enabled quick and interactive analysis of data. Missions were flown with GPSs, cameras, spectrometer and laser altimeter instruments. The combination of a relatively accurate elevation data set and rapid data analysis enabled verification of laser altimeter measurements and determining the altitude limits at which the laser altimeter became ineffective. It also provided a very simple way to compare the accuracy of the different onboard GPSs.

In September 2012, the system was used to visualize flights for an oil spill exercise outside Berlevåg, Norway. The ability to create image mosaics was developed during the campaign, highlighting one of the strengths of a fully custom visualization system. Due to a catastrophic failure with the power supply to the main image-processing computer, the crew was left without the capability to create image mosaics. By doing onsite development, the feature was incorporated into the visualization system and enabled delivery of image products despite the lack of high-performance compute resources. The visualization also served as a good way of communicating flight status to ground-based personnel not directly involved in flight operations.

7. Conclusion

This paper has presented Norut Geo Viz, a system for realtime and post-landing visualization of flights with unmanned aircraft. The system visualizes flights in 3D on a virtual globe, by combining several different raster and elevation map layers with flight and instrument data. The system has been used by Norut for flight campaigns conducted in Ny-Ålesund on Svalbard during summer 2012 and in an oil spill exercise near Berlevåg, Norway during fall 2012. The system has enabled operators to quickly analyze data collected by instruments such as cameras, spectrometers and lasers, both during flight and after landing. The system efficiently maps images to their approximate location on the ground, and presents additional measurements spatially in the visualization at the location where they were made. The system runs on Mac OS X, Linux and iOS.

Acknowledgments

This work was supported by the Norwegian Research Council (Arctic Technology program). Map data in screenshots are sourced from the Norwegian Mapping Authority

and the Norwegian Polar Institute.

References

Bourke, P. (1999), 3D stereo rendering using OpenGL (and GLUT). http://paulbourke.net/stereographics/stereorender/ (last visited April 25. 2013).

Donnelly, W. & Lauritzen, A. (2006), Variance shadow maps, *in* 'Proceedings of the Symposium on Interactive 3D Graphics and Games 2006', pp. 161–165.

Göddeke, D., Strzodka, R. & Turek, S. (2005), Accelerating double precision FEM simulations with GPUs, *in* F. Hülsemann, M. Kowarschik & U. Rüde, eds, '18th Symposium Simulationstechnique', Frontiers in Simulation, SCS Publishing House e.V., pp. 139–144. ASIM 2005.

Google, Inc. (2005), Google earth. http://earth.google.com/ (last visited April 24. 2013).

Hagen, T.-M. S. (2011), *Interactive Visualization on High-Resolution Tiled Display Walls with Network Accesible Compute- and Display-Resources*, University of Tromsø, Norway. Ph.D. dissertation.

Jensen, A. M. (2009), *gRAID: A Geospatial Real-Time Aerial Image Display for a Low-Cost Autonomous Multispectral Remote Sensing*, Utah State University, USA. Master thesis.

Kaltenbrunner, M., Bovermann, T., Bencina, R. & Costanza, E. (2005), TUIO - a protocol for table-top tangible user interfaces, *in* 'Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation'.

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B. & Huebner, R. (2003), Level of Detail for 3D Graphics, Morgan Kaufmann.

Meier, L. (2009), MAVLink Micro Air Vehicle Communication Protocol. http://qgroundcontrol.org/mavlink/start (last visited February 6. 2014).

NASA (2002), WorldWind. http://goworldwind.org/about/ (last visited April 24. 2013).

Ohlarik, D. (2008), Precisions, precisions. http://blogs.agi.com/insight3d/index.php/2008/09/03/precisions-precisions/ (last visited October 14. 2013).

Stöckl, R., Vermote, E., Saleous, N., Simmon, R. & Herring, D. (2005), The blue marble next generation - a true color earth dataset including seasonal dynamics from modis, *Published by the NASA Earth Observatory*.

Thorne, C. (2005), Using a floating origin to improve fidelity and performance of large, distributed virtual worlds, *in* Proceedings of Cyberworlds, IEEE Computer Society, pp. 263– 270.

U.S. Geological Survey (1996), Global 30-arc-second elevation data (GTOPO30).

U.S. Geological Survey (2006), Shuttle radar topography mission, SRTM v2, complete 3 arc second dataset.

Figure captions

Figure 1: Screenshot from Norut Geo Viz, visualizing a flight conducted near Berlevåg, Norway in 2012.

Figure 2: The overall system architecture of Norut's unmanned systems.

Figure 3: The level of detail (LOD) pyramid. Higher LODs cover the same area, but at increasingly higher resolutions.

Figure 4: The visualization's design.

Figure 5: Norut Geo Viz running on an iPad.

Figure 6: Spectrometer measurements from Ny-Ålesund on Svalbard.

Figure 7: A crop from two image mosaics created using images from the same flight.

Figure 8: The visualization running on an Archimedes Exhibitions SessionDesk 2.0 multi-touch table.

Figure 9: Realtime stitching of terrain meshes at different levels of detail.

Figure 10: Shadow casting in Norut Geo Viz. The model of the bridge is made by Bent Ebeltoft, and is available from the Trimble 3D warehouse. The box buildings are generated based on polygons extracted from the Norwegian Mapping Authority raster map layer.

Figure 11: Flight planning with Norut Geo Viz.

Figure 12: Histogram displaying distance between camera positions computed by NGV and corresponding camera positions computed by AgiSoft PhotoScan.

Figure 13: Histogram displaying the distribution of distances between corresponding corner and center points of images mapped to the ground as computed by NGV and PhotoScan.

Figure 14: Comparison of image mapped by NGV vs image mapped by PhotoScan. Maximum distance between corresponding corners in this image is 622.9 meters.

Figure 15: An image where the mapping done by NGV better matches the “ground truth” established by PhotoScan. Here, the maximum distance between corresponding corners is 14.0 meters.